Optical Graphical Recognition via Deep Learning

Oscar Lo and Runying Chen
University of Washington
Seattle, WA 98195
olo126@uw.edu, rc76@uw.edu

Abstract

Image captioning has been done well these years via some outstanding models. [5, 6] These models provide a script of text to describe the content of an image. Yet, there are few models designed for describing images of graphs. For some optical graph recognition models describing images of graphs, they encounter the problem of edge crossing. [1] This paper tries to design a model to describe a graph image and print the corresponding representation. The designed model uses a residual network with 50 layers (ResNet50) [3] to extract features from graph images, a visual projection layer and a transformer network to do captioning.

1. Introduction

This paper works on the problem of optical graph recognition. Optical graph recognition is concerned with recognizing the topology and structure of a graph, given an image or drawing. This requires identifying the distinct edges and nodes shown in the image of the graph, along with correctly matching the labels of the nodes and edges. Finally, it is necessary to connect the nodes and edges in a text representation of the graph in the same way as shown in the input image. Graphs are a common and widely used way of conveying information. There are a variety of fields that utilize graphs such as using graphs to represent social networks, chemical compounds, or financial networks. While the purpose of graphs is to allow for easily and quickly understandable representations of data, they must still be turned into a textual representation when fed to computers. If able, this will allow hand-drawn graphs and the like to be easily converted to use in programs. This is a challenging process that has not previously received much attention.

1.1. Proposal

In this paper, a model is proposed to try to address this issue. An object detection R-CNN is expected to be good at identifying the parts of a graph [8], which should be the best choice to extract nodes and edges from graph images for our model. However, due to lack of proper data sets that fit our needs, we are unable to train an R-CNN. Hence, we use ResNet50 to extract features of graph images and combine it with a transformer to read the images and to output dictionaries with nodes as keys and lists of edges as values. There have been many previous works that use neural networks to attempt to work with graphs, but the vast majority are like [10] where a text representation is taken and passed through a neural network. We seek to attempt to instead pass an image and get the representation that these models would usually use at input. In this way, we hope to help bridge the gap between hand drawn graphs and those usable in programs.

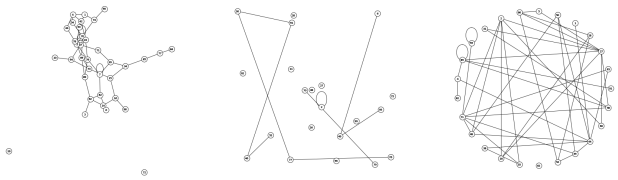


Figure 1. (a) shows the case of nodes overlapping. Part of information of the graph is covered. (b) shows the case of edges crossing irrelevant nodes. In the center of the graph, the self-pointing nodes is passed by an edge that is not supposed to connect the self-pointing node. (c) shows the case of edges crossing.

1.2. Contribution

In this paper, we find that denoting graphs via images with our model is not a good idea as it exposes several issues. First, graph images are inefficient and expensive. In our data sets, the graphs in images have 40 nodes and 80 edges in maximum, while the image size is 512*512*3. For an adjacency matrix, such a graph is in size of 40*40. Second, graph images introduce problems like nodes overlapping and edges crossing irrelevant nodes in addition to prior identified issues like edges crossing. Fig.1 shows a typical example.

We apply several layout strategies to avoid nodes overlapping (unable for the model to parse or represent) and edges crossing irrelevant nodes (difficult to distinguish if the nodes and edges are connected). We keep the edges crossing problem because there is not layout can handle this problem and it is common. Our model is expected to handle this problem well. Also, different from other image captioning models, our model is forced to read more details in graph images in order to generate a correct representation of the input graph image. Our findings show that current vision-language models, have the capability to track general structure in images of graphs and transfer that knowledge into a text representation but struggle to go any farther. It also shows that while current machine learning methods are more than capable of parsing standard images of dogs, cars, and other objects, extracting graph representations themselves from provided drawings of graphs is still out of reach. However, if the ability to only generally track graph structure and patterns is properly expanded upon, it still possibly holds the potential to greatly simplify workflows involving graphs taken from the real world. Furthermore, while our model's capability in parsing standard graphs is limited, there are paths to improvement and we believe that a properly from-scratch model trained specifically on graphs implies the potential for improving utility in analysing other graph-like images such as maps or structural engineering.

2. Related Work

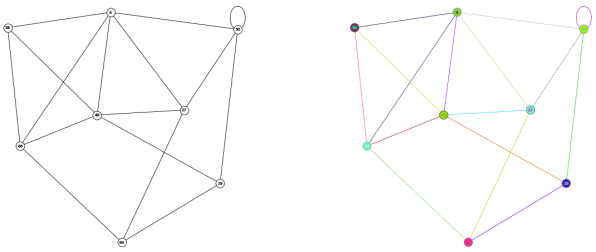
One of the few papers that also tackled the optical graph recognition problem is [1], which uses image processing techniques such as image segmentation and skeletonization to classify nodes and edges. One of the areas where the model had the most difficulties was when edges of the image cross, which in [1] splits the graph's edges into edge sections, which are later merged together through some heuristics to reconstruct the true edges of the graph. The image processing algorithm discussed in [1] does particularly poorly on graphs with many crossings, with up to 77.25% of false positives (a pair of nodes falsely identified as having an edge between them) occurring in graph drawings with many edge crossings in small areas. Another similar area of difficulty was identified as graphs with only a slight angle separating two edges. One related piece of work has also been done to identify the graph structure of chemical compounds from drawings of molecules using deep learning [7]. A common pattern arises in that they too, uses image segmentation to identify the atoms and bonds in the graphs. They then use convolutional networks to classify the atoms and bonds. However, [7] focuses solely on chemical graphs and properties meaning that their work cannot be generalized to our goal of creating an optical graph recognition model for any graph in general. A large part of [7] focuses on identifying types of atoms and bonds, much of which is not relevant to graphs observed outside of very specific settings such as chemistry. In the same vein, we have found little research that directly applies a deep learning-based approach to perform optical graph recognition on graphs in general rather than specialized use cases. In this paper, we will experiment with a combined model of ResNet50 model and a transformer model to approach this problem, taking images of undirected graphs with labeled nodes as input and producing a representation akin to a dictionary, with nodes as keys and lists of edges as values, as output. (Adjacency matrices and GraphML file are also popular means to represent a graph. These three means can transformed from one to another.) Of the few works that attempt the problem of optical graph recognition, [2] demonstrates the utility of transformers in analyzing sub-sections on graphs, especially when combined with features like self-attention mechanisms. However, while [2] focuses on extracting the sub-graphs and sub-graph representations of input graphs, we intend to go a step further, breaking input graphs down to an atomic level to extract edges and nodes that will be used by our transformer to build the previously mentioned graph representation. Finally, there has been much work like [10] which attempt the similar problem of feeding a graph representation rather than a graph image to a model, then performing graph classification and other such tasks on the input. Our work will aim to create a model that can be used in the pipeline to generate from images of graphs a representation that can be fed into GNNs and other such models.

3. Methods

The primary objective of this report is to generate a graph representation in the form of an adjacency matrix, which can be used as an input for GNN models. Currently, most image captioning models provide textual descriptions for a great number of images [5, 6], but few models are designed to read an image of a graph and describe the image in a specific form. In this report, we view the dictionary as a form of language with defined syntax, which allows us to describe graph images, so that it is guaranteed that a graph can be uniquely represented even though it has different layouts. We have learnt that the transformer architecture achieves great success in most image captioning models, so this architecture is our first choice of the main part of our expected image captioning model.

3.1. Data Generation

To address the absence of a certain dataset for images of graphs, we have to create our own dataset. We apply Python package networkx and matplotlib, producing graph images based on randomly generated non-negative integers, which served as nodes, and lists of tuple containing arbitrary chosen number from those non-negative integers. Limited to



(a) (b)

Figure 2. (a) shows the black-and-white example. (b) shows the colored example. Both examples are in spring layout and have the same dictionary representation, which is {4: [(4, 27), (4, 48), (4, 50), (4, 58), (4, 66)], 19: [(19, 48), (19, 50), (19, 64)], 27: [(4, 27), (27, 48), (27, 50), (27, 64)], 48: [(4, 48), (19, 48), (27, 48), (48, 58), (48, 66)], 50: [(4, 50), (19, 50), (27, 50), (50, 50)], 58: [(4, 58), (48, 58), (58, 66)], 64: [(19, 64), (27, 64), (64, 66)], 66: [(4, 66), (48, 66), (58, 66), (64, 66)]}

the image size, which is 512*512*3 to avoid blurry scenes, there are at most 40 nodes and 80 edges for a single graph image. In order to prevent nodes overlapping, edges crossing irrelevant nodes, we apply pygraphviz layout, Kamada-Kawai layout, circular layout, and spring layout. They also serve as data augmentation since for the same graph, different image representations are produced. We also randomize the color of nodes, nodes' borders, edges, and labels for data diversity as well as data augmentation. The images serve as inputs, and the outputs are the corresponding dictionary representations. On purpose of uniquely denoting a graph, we have defined syntax for the dictionary representation. That is, the nodes as keys in the dictionary should be in ascending order. Plus, edges are denoted as tuples. The values to the keys in the dictionary are lists of edges. For edges in a list, edges are arranged in ascending order of the first element in the edge tuple. Fig.2 shows a black-and-white example and a colored example of the data.

3.2. Model Architecture

Our image captioning model for graphs is designed to generate dictionary representations, which can be transformed into adjacency matrix for GNN models. The architecture follows the example of the transformer parts of successful image captioning models [5, 6] as well as having the encoder and decoder changed in detail uniquely for graph characteristics.

3.2.1 Preprocessing

Instances are shuffled and retrieved from the dataset, and the vocabulary for captions are non-negative integers from 0 to 99, along with some characters for dictionary form. Hence, the vocabulary size is 110.

3.2.2 Image Encoding Layer

For the image encoding component in our model, we leverage a pre-trained ResNet50, which is able to extract high-level and abstract features from input images. [3] The pre-trained ResNet50 is implemented via PyTorch, which is loaded and integrated into our overall architecture. We also train this pre-trained ResNet50 specifically with our data in training.

3.2.3 Positional Encoding Layer

To incorporate spatial information into the model, a positional encoding layer is applied. This enables the model to consider the spatial arrangement of nodes and edges within the graph images. In is report, we apply the same positional encoding technique as the one used in *Attention Is All You Need* [9].

3.2.4 Visual Projection Layer

We add a visual projection layer mainly for the image captioning task. This layer adapts features learned by the ResNet50 backbone, and resizes of the output from the ResNet50 layer via a linear layer.

3.2.5 Transformer Layer

Since the transformer architecture [9] is widely successful in image captioning models, our design integrates a transformer layer. This layer processes the enhanced input, capturing contextual relationships and dependencies among nodes and edges for generating a meaningful graph representation. An ordinary transformer layer includes "word to index" for generating texts, but our model prints dictionary representations in the end; thus, we use a list of non-negative integer (number of nodes) and characters of constructing a dictionary to replace the list of word to index. The final output is producing an dictionary representation that serves as the graph representation. This representation can be transformed into adjacency matrix, intending to be seamlessly compatible with GNN models for downstream applications.

3.3. Training Procedure

The model is trained using the generated dataset with appropriate splits for training, validation, and testing. The model can be trained by back propagation. The optimizer is Adam. [4] We compare the printing dictionary representation of our model to the ground truth in the original dataset via softmax loss function. The learning rate of our model is 5e-4.

3.4. Challenges

One challenge in building this model is that graph images we used are generated by computers and there are no similar graph images can be found at CIFAR-10, which is the dataset used to train ResNet50. Thus, we are unable to guarantee the effectiveness of feature representations produced by ResNet50 on our dataset. Also, an image in our dataset is 512*512*3, which is much larger in size than any image in CIFAR-10.

Also, the captioning part in our model is over 1000 for maximum, which is much larger than ordinary image captioning tasks. This induces longer training time.

4. Experiments

4.1. Setup

Our dataset has 4120 instances in total. Half are black-and-white instances and the other half are colored instances. 2000 instances are in pygraphviz layout, 20 instances are in Kamada-Kawai layout, 100 instances are in circular layout, and the last 2000 instances are in spring layout. Kamada-Kawai layout has effect on dealing with nodes overlapping and edges crossing irrelevant nodes, but not much. Thus, we generate few instances in this layout for data augmentation, which allows us to check manually if the instances suitable for training the model. Circular layout is good at solving nodes overlapping and edges crossing irrelevant nodes, but the generated graphs have same shape (that is, nodes are arranged in a circle), which has opposite impact on generalizing the model. Pygraphviz layout and spring layout work best on generating graph images, so they cover most of instances in the dataset.

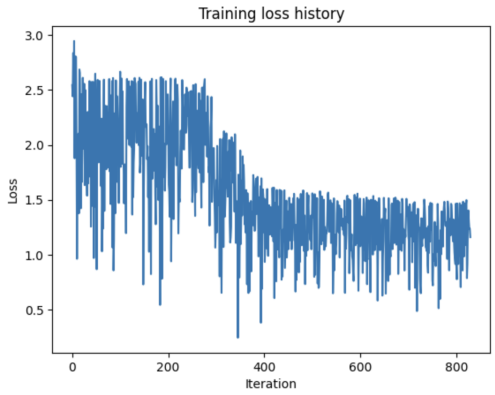
The experiment is conducted on Colab with T4 GPU because each single data is large and the length of captions produced is long, which is 1397 at maximum.

4.2. Results

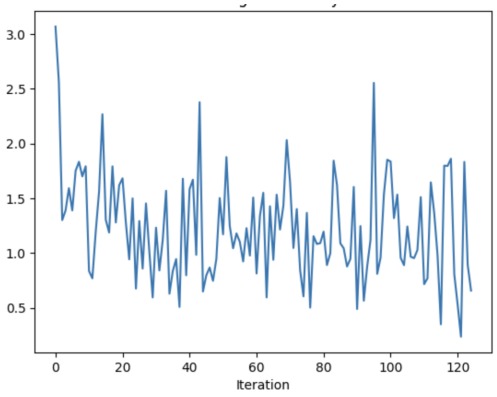
In our experiments, we trained our model by passing the converting the images of a graph into a 3 channel matrix and feeding that to our ResNet50-transformer model. We have the model produce a string output where a graph representation in the form of a dictionary is used. The nodes in a graph are set as the keys and the edges that contain the key are the values of the dictionary. We split the data into a $80/10/10$ train/validation/test split. As can be seen from Fig.3, the results of our training was noisier than expected. While loss did train downward, the loss would see large fluctuations from values as small as 0.1 to more than 1. We believe this is likely due to the size of our data which necessitated the use of rather small batches to train our model. We also believe that the model architecture could be improved as discussed later. Additionally our validation and test accuracy were lower than initial expectations, hovering at around 12%. We believe that this is partially caused by the harshness of our judgement criteria. When comparing the two string outputs, an output was only counted correct if it was the exact same as the ground truth caption representation. This did not take into account representations where only slight errors occurred or where orders of nodes or edges may have been switched but still defined the same graph. In future work, a more flexible and sensitive method of evaluating the correctness of caption graph representations is advised, such as converting the caption string to a data structure that can easily be used to check not just equivalence without regards to node or edge order, but also measure the level of incorrectness in a caption. This would help provide a more accurate image of model capabilities. To attempt to at least partially address this issue, we randomly sampled 50 test generations of our model to compare manually with the ground truth labels. We found that while our model could fairly accurately parse graphs of around 10 nodes or less, performance, expectedly, decreased as graph complexity increased. At most, the model was able to capture an accurate semblance of a graph through general patterns and structure when graph images were generated from model output captions. We believe that a portion of the cause of the model's inaccuracy is due to using a pre-trained ResNet50. ResNet50 was trained on ImageNet which contains image types of real world objects lacking the detail that complex graphs are expected to have. We also attempted other models for the step of graph feature extraction such as a pre-trained faster-RCNN ResNet50 for bounding boxes. However, after testing on a sample of graph images, we found that, similar to our model, the pre-trained faster-RCNN ResNet50 could not keep up with the level of details in our graphs. Thus, we believe that as is, training a model to produce a graph representation caption from a graph image is unfeasible, but possibly training a model from scratch to perform image recognition on graph images would help improve feature extraction and subsequently the performance of the model.

5. Discussion

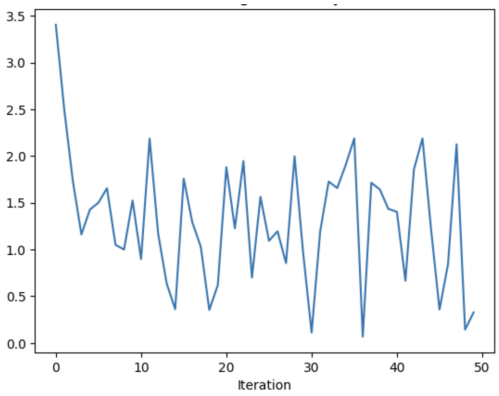
The difficulty of training such a model shown above prompts a critical concern on the feasibility of representing graphs through images. This is because an image with size as 512 * 512 * 3 can only denotes a graph with 40 nodes and 80 edges in total (in our case). However, a graph with hundreds of nodes and edges can be denoted with an adjacency matrix using smaller memory space. This scalability problem on the image-based approach can be more severe if a large graph is required to denote. Also, typically, a graph used for GNN models is larger than what we trained on our model. Hence, the smaller graphs generated by our model might struggle to generate effective results for GNN applications.



(a)



(b)



(c)

Figure 3. (Example of training loss graph over 50 to 800 iterations using learning rates in range from 0.01 to 0.0005 and Adam optimizer

Yet, diverging from conventional most image captioning tasks, a positive aspect emerges from this paper. Through a utilization of a specific group of images and captions, coupled with a self-defined syntax to formalize captions, an image captioning task is still approachable, indicating there is more can be done with image captioning models.

References

[1] Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Josef Reislhuber. Optical graph recognition. *Journal of Graph Algorithms and Applications*, 17(4):541–565, 2013. 1, 2

[2] Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning, 2022. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1, 3

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 3

[5] Chenliang Li, Haiyang Xu, Junfeng Tian, Wei Wang, Ming Yan, Bin Bi, Jiabo Ye, Hehong Chen, Guohai Xu, Zheng Cao, Ji Zhang, Songfang Huang, Fei Huang, Jingren Zhou, and Luo Si. mplug: Effective and efficient vision-language learning by cross-modal skip-connections, 2022. 1, 2, 3

[6] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023. 1, 2, 3

[7] Martijn Oldenhof, Adam Arany, Yves Moreau, and Jaak Simm. ChemGrapher: Optical graph recognition of chemical compounds by deep learning. *Journal of Chemical Information and Modeling*, 60(10):4506–4517, sep 2020. 2

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 1

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 3

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019. 1, 2